IN THE SPECIFICATION

Please amend the specification as follows:

The paragraph beginning at page 1, line 10, is amended as follows:

In the rapid development of computers many advancements have been seen in the areas of processor speed, throughput, communications, and fault tolerance. Initially computer systems were standalone devices in which a processor, memory and peripheral devices all communicated through a single bus. Later, in order to improve performance, several processors [[and]] were interconnected to memory and peripherals using one or more buses. In addition, separate computer systems were linked together through different communications mechanisms such as, shared memory, serial and parallel ports, local area networks (LAN) and wide area networks (WAN). However, these mechanisms have proven to be relatively slow and subject to interruptions and failures when a critical communications component fails.

The paragraph beginning at page 1, line 20, is amended as follows:

However, no matter what type of architecture is utilized, the classic problem found in communications is determining if the receiver has the capability of receiving an entire message at any given moment in time and that the entire message was actually received by a particular receiver. One method utilized in the past to assure a sender that the receiver has the capacity to receive messages involved the transmission of tokens from the receiver to the sender. This mechanism was known as an absolute credit update method. In this absolute credit update method the potential receiver <u>of a</u> message would, for example, transmit three tokens to a potential sender. These three tokens would indicate that the receiver is able to receive three packets or chunks of data representing one or more messages. The sender would then send these three pieces of information to the receiver. Thereafter, the receiver would then transmit additional tokens indicating the ability to receive additional information.

The paragraph beginning at page 2, line 10, is amended as follows:

The problem encountered <u>in</u> utilizing this absolute credit method was that a highly reliable communications fabric or network is assumed to exist. No provision exists in the absolute credit update method for detection of lost packets or pieces of information or messages. However, as previously discussed, failures in hardware occur and information transmitted over a network may be lost in the transmission process. Mechanisms do exist for lost packets recovery. However, these mechanisms very often are complex and utilize a significant amount of the communications bandwidth available simply for detection of lost information. Therefore, the existing mechanisms for lost information detection and recovery do not allow a network to fully utilize the bandwidth capability available to it.

The paragraph beginning at page 2, line 20, is amended as follows:

Therefore, what is needed is a method and computer program that will determine if a potential receiver of a message has the capability of receiving that message at a given point in time. Further, this method and computer program must be able to detect when a [[lost]] message or data <u>loss</u> has occurred. This method and computer program for detection of a lost message must be simple to implement and execute and utilize minimum processor time as well as communications bandwidth. Therefore, this method and computer program must be able to keep communications speed as close to the bandwidth limit as possible.

The paragraph beginning at page 7, line 9, is amended as follows:

Still referring to FIG. 2, communications between the HCA 60 and TCA 80 in I/O units 310 would occur over the InfiniBand fabric 300. The InfiniBand fabric 300 would be either switches 50 or a direct serial connection between HCA 60 and TCA 80. Besides TCA 80 I/O units 310 would also include a corresponding flow control module 260 in which the embodiments of the present invention may reside but are not limited thereto. Further, I/O units 310 would include I/O controller 70 which would interface to I/O device interface 320. I/O device interface 320 would include, but <u>be</u> not limited to, items such as mass storage controllers and other parallel or serial communications interfaces. In turn, I/O device interface 320 may interface to mass storage devices 330 or an outside LAN or WAN network 340.

The paragraph beginning at page 7, line 19, is amended as follows:

FIG. 3 is an example of a flow control message header 365 used in the example embodiments of the present invention. This flow control message header 365 may be transmitted with applications send messages or as a separate message. The flow control message header 365 has two components. The first component ~~components~~ is the message sent 360 which indicates the total number of messages sent by the initiating or transmitting processor. The second component is the message limit 370 which represents the total number of messages that the remote node may send over the connection. As will become evident from the discussion of FIGS. 4-9 by utilizing message sent 360 and message limit 370 it is possible to guarantee that a receiving processor of a message will have required message buffer to store the message. Further, utilizing message sent 360 and message limit 370 it would be possible to detect lost messages or pieces of data.

The paragraph beginning at page 9, line 12, is amended as follows:

FIG. 5 is an example flowchart for the Get Credit module used in the example embodiments of the present invention. The get credit module begins execution in operation 500 and immediately proceeds to operation 510. In operation 510, it is determined whether the variable send limit is greater than the variable send count incremented by one. The variable send limit is the maximum number of messages that may be sent to a remote receiving queue in a remote processor. Further, the variable send limit would include the number of messages sent by the transmitting node plus the number of received buffers still available in the remote receiving processor node. If the variable send limit is greater than the variable send count incremented by one then processing proceeds to operation 540. In operation 540 the variable get credit is set to true. Thereafter, processing proceeds operation 550 where processing terminates.

The paragraph beginning at page 10, line 1, is amended as follows:

Still referring to FIG. 5, if the variable send limit is less than or equal to the variable send count incremented by one then processing proceeds to operation 520. In operation 520, it is determined if the variable send limit is equal to the variable send count plus one and that the variable new credits is greater than zero. If the variable send limit is equal to the variable send count plus one and the variable new credits is greater than zero then processing proceeds to operation 540 as previously discussed. However, if the variable send limit is not equal to the variable send count plus one and new credits is not greater than zero then processing proceeds operation 530. In operation 530, the variable get credit is set false and processing proceeds to operation 550 where processing terminates.

The paragraph beginning at page 10, line 11, is amended as follows:

FIG. 6 is an example flowchart for the Periodic Update module used in the example embodiments of the present invention. The periodic update module begins execution in operation 600 and immediately proceeds to operation 610. In operation 610 the variable send count is incremented by one. Thereafter, in operation 620 message sent 360 field of the flow control message header 365 is incremented by one. In operation 630 variable available credits is incremented by the value contained in the variable new credits. Thereafter, in operation 640 the message limit 370 field of the flow control message header 365 is set equal to the variable consumed credits plus the variable available credits. The variable consumed credits is the total number of messages that were either received or lost. The variable consumed credits is initially set to zero during a connection session and is incremented for each received or dropped message. Further, the variable consumed credits [[is]] in each is used to update the message limit field 370 of the flow control message header 365. In operation 650 the variable new credits is set to zero and processing terminates in operation 660.

The paragraph beginning at page 11, line 3, is amended as follows:

FIG. 7 is an example flowchart for the Receive Done module used in the example embodiments of the present invention. The Receive Done module begins execution in operation 700 and immediately proceeds to operation 705. In operation 705, the variable consumed credits

is incremented by one. Thereafter, in operation 710 the variable available credits is incremented by one. In operation 715, it is determined whether the message sent 360 field of the flow control message header 365 is greater than the value contained in the variable consumed credits. If in operation 715 it is determined that the value contained in the message sent 360 field is not greater than the value contained in the variable consumed credits then processing proceeds to operation 750. Branching to operation 750 would indicate that no message or piece of data was dropped during transmission. In operation 750 the periodic update timer is reset. This periodic update timer serves to prevent blockages from forming between communicating parties where [[we]] due to some failure or error a transmitting party has run out of credits. The periodic update timer will be further discussed in reference to periodic update timer module illustrated in FIG. 9.

The paragraph beginning at page 11, line 19, is amended as follows:

However, if in operation 715 it is determined that the message sent 360 field in the flow control message header 365 is ~~less~~ greater than or equal to the value contained in the variable consumed credits then processing proceeds to operation 720 where it is presumed that a message or piece of data has been dropped during transmission. In operation 725, a variable drop count is set equal to the message sent 360 in the field flow control message header 365 less the variable consumed credits. The variable consumed credit is the total amount of messages received or dropped. This consumed credits variable is initialized to zero when a communications connection is established. Thereafter, in operation 735 it is determined if the variable drop count is greater than the variable available credits. As previously discussed, the variable available credits is the number of messages or space available on a received work queue that have been allocated for a transmitting processor to send messages to it. If the value contained in the variable drop count is greater than the value contained in the variable available credits then processing proceeds to operation 740. In operation 740 the variable new credits is incremented by the variable available credits. Thereafter, processing proceeds to operation 745 with the variable available credits [[is]] set to zero. Processing then proceeds to operation 765 where the new credit information is updated. In operation 770, a variable send limit is set equal to the message limit 370 in the flow control message header 365. In operation 775, processing of any

pending send request then proceeds. Thereafter, in operation 780 the threshold module is then activated and thereafter processing terminates in operation 785.

The paragraph beginning at page 12, line 17, is amended as follows:

Still referring to FIG. 7, if in operation 715 it is determined that the message sent to 360 field contains a value that is not greater than the variable consumed credits then processing proceeds to operation 750 as previously discussed. In operation 750, the periodic update timer is reset. Thereafter, processing proceeds to operation ~~operates~~ 765 as previously discussed.

The paragraph beginning at page 12, line 22, is amended as follows:

Still referring to FIG. 7, if in operation 735 it is determined that the variable drop count has a value which is equal to or less than the variable available credits then processing proceeds to operation 755. In operation 755 the variable available credits is decremented by the value contained in the variable drop count. Thereafter, processing proceeds to operation 760. In operation 760, the variable new credits ~~credit~~ is incremented by the variable drop credits. Processing then proceeds to operation 765 as previously discussed.

The paragraph beginning at page 13, line 5, is amended as follows:

FIG. 8 is an example flowchart for the Post Receive module used in the example embodiments of the present invention. The Post Receive module begins execution in operation 800 and immediately proceeds to operation 810. In operation 810 the variable new credits is incremented by one. In operation 820 any pending requests ~~request~~ for messages are processed. Thereafter, in operation 830 processing terminates.

The paragraph beginning at page 13, line 11, is amended as follows:

FIG. 9 is an example flowchart for the Threshold Check module used in the example embodiments of the present invention. The Threshold Check module begins execution in operation 900 and immediately proceeds to operation 910. In operation 910 it is determined whether the variable available credits is less than [[then]] the value contained in a variable credit threshold. The variable credit threshold is a counter used to limit the number of credit updates

and ensures that credit updates are provided for unidirectional traffic patterns. If the value contained in the variable available credits is greater than or equal to the value contained in the variable credits threshold then processing proceeds to operation 930 as will be discussed in further detail ~~detailed~~ ahead. However, if the value contained in the variable available credits is less than the value contained in the variable credit threshold then processing proceeds to operation 920. In operation 920 the Post Send module, discussed in reference to FIG. 4, is executed so that a credit message is sent to the transmitting node or processor in order to update the number of credits available to that processor. Thereafter, processing proceeds to operation 930 where the periodic update timer module, previously discussed, is reset. Processing then terminates in operation 940.